0325.00239
CD99043

METHOD AND APPARATUS FOR AUTOMATED ENUMERATION, SIMULATION, IDENTIFICATION AND/OR IRRADIATION OF DEVICE ATTRIBUTES

Field of the Invention

The present invention relates to automated enumeration generally and, more particularly, to a method, software and/or apparatus for automated enumeration, simulation identification and/or irradiation of device attributes.

Background of the Invention

Conventional methods exist to automate enumeration of all fuse locations on a die. The conventional methods do not associate fuse locations with a schematic path and/or a verilog path. Conventional methods exist to manually associate a fuse path to a fuse location or the fuse location to the fuse path, one at a time. The conventional methods to manually associate the fuse path to the fuse locations, or vise versa, use a layout versus schematic (LVS) cross-probe user-interface. Conventional verilog simulation paths are derived by manual translation of schematic paths aided by visual inspection of a netlist.

Additionally, conventional methods do not effectively collect thorough and accurate fuse path versus fuse location data.

The manual LVS cross-probe cannot process the fuse path versus fuse location data for large numbers of devices in a timely, cost-effective manner. Without a thorough and accurate path versus location data, methods to verify repair programs and redundancy documentation are tedious and error prone.

Summary of the Invention

The present invention concerns a method of automated enumeration of one or more devices comprising the steps of (A) generating an enumeration of a plurality of fuses and (B) compiling data for each one of said plurality of fuses, wherein the data comprises (i) one or more schematic path data, (ii) one or more simulation path data and/or (iii) one or more physical location data.

The objects, features and advantages of the present invention include providing a method and/or apparatus that may provide (i) automatic enumeration of all fuses in the design and collection of a schematic path and annotated properties for each fuse, (ii) automatic determination of a verilog simulation netlist path for each fuse, (iii) automatic determination of a physical location for each fuse, (iv) automatic verification of fuse

locations against additional identifying shapes drawn in the layout, (v) coordinates, such as from repair program output, are automatically translated to verilog programming statements (vi) automatic and/or manual ad-hoc queries and searches to isolate groups of fuses for tabular listings or to program them in verilog, (vii) automatic translation of existing program statements to their physical locations, (viii) manual annotation of the schematic hierarchy at multiple levels with descriptive fuse properties, and/or (ix) manual searches or look-ups based on expression matching against this description property.

Additionally, fuse data is generally enumerated and collected from the design flow into a separate file. The file may thereafter be used independently of the original design flow data and tools. Traditional access to cross-probing functionality of simulation netlisters and LVS tools may require simultaneous availability of (i) original design data, (ii) specific version of vendor software, (iii) specific architecture of host computer, and (iv) licenses to enable the vendor software.

Brief Description of the Drawings

These and other objects, features and advantages of the present invention will be apparent from the following detailed description and the appended claims and drawings in which:

FIG. 1 is a block diagram of a preferred embodiment of the present invention;

FIG. 2 is an alternate block diagram of the present invention; and

FIG. 3 is another alternate block diagram of the present invention.

Detailed Description of the Preferred Embodiments

Referring to FIG. 1, a block diagram of a method 100 is shown in accordance with a preferred embodiment of the present invention. Fuses in a design may be enumerated and data collected and stored in a file. The data may include schematic path data, verilog simulation path data and/or physical fuse location data. Multiple representations of the design and/or data may be accessible by other tools, such as netlister and/or layout versus schematic (LVS). The data may be used to enumerate the fuses. The data may be implemented in order to test and/or repair the design.

References to "verilog" refer generally to the verilog hardware description language (HDL) as defined by the IEEE 1364-1995 standard.

The file may be implemented to answer look-ups and translations as directed by an operator or apparatus. Automated programming of fuses in a design simulation may verify (i) accuracy of a laser repair program and/or (ii) design redundancy functionality. The file may also be implemented to ensure accurate documentation of redundancy methodology for the design. The look-ups and translations may be constrained to include and/or exclude fuses based on (i) physical location range specifications, and/or (ii) exact or pattern matching of (a) schematic paths, (b) verilog paths and/or (c) description properties.

The method 100 may comprise a design data block 102, a netlist block 104, a simulation block 106, a generation block 108, a table block 110, an application block 112, a program statement block 114, a location block 116, a schematic/simulation block 118, a repair block 120 and a test block 122. The generation block 108 may receive fuse data from the design data block 102. The generation block 108 may write the fuse data into a file and perform error checking on the file.

The generation block 108 may perform enumeration of fuses and the collection (compilation) of data for each fuse. The schematic design data may comprise (i) schematic path data and (ii) property data formed from hierarchical contributions. For each fuse, the generation block 108 may generate verilog path data. The generation block 108 may generate the verilog path data depending on the methodology implemented in netlist block 104. The generation block 108 may generate the verilog path data using either a first or a second method.

The first method may translate the schematic path data to the verilog path data via dead-reckoning. The first method may implement the same algorithm known to be used by the netlist block 104. The netlist block 104 may be implemented as FNL-based netlisters, HNL-based netlisters, or any other type netlister in order to meet the criteria of a particular implementation. The second method may map the schematic path data to the verilog path data by direct lookup using netlister map files that may be implemented within the netlist block 104. An associated netlister API (application-programming-interface) may be represented in the block 102. The netlister map files and the netlister API may be implemented within the netlist block 104. If an operator or

apparatus requests a non-LVS mode (for example if clean LVS is not available yet), the fuse generation block 108 may write the fuse data collected to a file and terminate.

The LVS data is generally used to collect the physical location data for each schematic path. LVS tools may access the physical location data in one of two methods. The first method may comprise a LVS tool that may provide (i) one or more APIs or (ii) one or more cross-referenced output files. The schematic path is generally translated from the files to one or more matching x/y device locations through one or more steps. The second method may comprise a LVS tool that may provide (i) an API or (ii) cross-referenced output files. The second method may translate a device x/y location into one or more matching schematic device paths.

When the LVS data is available, the data is generally used to extract partial information of which fuses may be electrically connected in parallel. The information is generally recorded to warn an operator or apparatus in the event of a request to program a strict subset of such a parallel group.

When the LVS data is available and is requested by an operator or apparatus, fuse locations given by the LVS may be compared against additional drawn layout shapes. If differences

are found in the fuse locations, a heuristic is generally used to attempt to matchup the differences. The heuristic may report the differences in a user-friendly manner to help an operator make layout corrections if desired. For example the LVS and the drawn layout may report different fuse locations. The fuse locations may be closer to each other than to any other unmatched locations. The close fuse locations may be listed together as probable intended matches. The generator block 108 may write the fuse data collected to the file and terminate. The generation block 108 may present the file to the table 110. The fuse data for each fuse of the device may be stored in the table 110.

The application block 112 may receive constraints and options given by the operator or another apparatus. The application block 112 may read each fuse data stored in the table 110 while applying the constraints to decide whether to retain or discard the fuse data.

The application block 112 may present one or more matching fuses in a first or a second format. The first format may present a tabular listing to the location block 116. The second format may present verilog programming statements to the program statements block 114. Options support how to sort fuses prior to

output. The application block 112 may control the order in which fields may be presented in the tabular method. The application block 112 may name a verilog path name prefix to be applied to the verilog paths, in order to program a device nested within one or more test-bench modules.

If the one or more matching fuses only partially represent a group of electrically parallel fuses (as indicated by grouping information stored within the fuse database file), warnings may be generated.

Construction of the repair block 120 may rely on part-specific redundancy information and errors found after a first-silicon delay part production. The repair block 120 may receive the errors from the defect block 122. The repair block 120 may be exercised in advance of the first-silicon for specific part failures. The repair block 120 may predict fuse locations that, if programmed, may correct a part experiencing failure. The method 100 may provide an easy and reliable method to perform simulations that emulate a design as if those locations were programmed on the die by the laser.

Referring to FIG. 2, an alternate method 200 of the present invention is shown. The method 200 may comprise a design

flow block 202 and a stand-alone block 204. The design flow block 202 may comprise a fuse network block 206, a fuse LVS block 208 and a design/database circuit 210. The fuse network block 206 and the fuse LVS block 208 may provide data to the stand-alone block 204. In one example the design/database circuit 210 may be implemented as a design flow and Opus design database.

The fuse network block 206 may collect data relevant to (i) schematic paths, (ii) properties, (iii) hierarchy and/or (iv) verilog paths. The fuse LVS block 208 may collect data relevant to (i) layout locations, (ii) parallel fuses, and/or (iii) LVS cross-reference to schematic paths.

The fuse network block 206 and the fuse LVS block may be implemented to drive applications within the stand-alone circuit 204. The stand-alone circuit 204 may comprise a fuse applications block 220. The fuse application block 220 may utilize either or both the fuse network block 206 and/or the fuse LVS block 208. The fuse application block may determine what data must be accessed and when the data will be accessed (e.g., within a software design tool with or without the flow schematic data only or schematic data plus layout data).

The fuse application block 220 may write one or more ASCII report files. A repair memo may be manually assembled from the report files. One or more steps may be manually determined and may be incorporated into the repair memo. The repair memo may be exercised to predict coordinates to program. The coordinates may be mapped to verilog paths. Simulations of the verilog paths may be performed to verify the expected function.

The design flow block 202 may further comprise a browse block 240. The browse block 240 may capture, in the schematic, enough knowledge of the fuses to elevate the fuses to a higher level of abstraction. The user may describe, in application terms, the desired redundancy event. For example, the redundancy event may comprise of replacing column C in quadrant Q. The coordinates to program may be automatically determined. However, the simulation may still be required to verify the function (e.g., to verify that the application specific knowledge captured in the schematic describing specific intent of each fuse is correct).

The Vampire LVS database may be used to generate a fuse database (e.g., fuse.fdb). The fuseGen may be run in the following ways (i) as a side effect of Vampire LVS, (ii) by invoking a skill function from the CIW, and/or (iii) from the command line. The

fuseGen may also be statically configured (e.g., using trf variables) to run for a set of specified cells. The fuseGen may also be run on a design tool schematic database (e.g., Opus) to get schematic instance paths, verilog paths and info strings for each fuse in the schematic hierarchy. The FuseGen may also be configured to run during verilog netlisting.

A file fuse.txt may be a text file listing all fuse co-ordinates. User inputs may be provided to fuseapp to output only certain groups of fuses (based on criteria defined in the user inputs). The fuse.txt file (or other text file(s) generated via fuseapp) may be included in the fuse repair document.

Referring to FIG. 3 a flow chart 300 of the operation of the method 100 is shown. The flow chart 300 may describe the procedure for using the present invention. The flow chart 300 may comprise a generate block 302, a repair memo block 304, a repair program 306, a test simulate bock 308, a feedback block 310, a map block 312, a logic simulation block 314, a SIMS check block 316, an error program check block 318 and an error memo check block 320.

The generate block 302 may generate a list of layout co-ordinates and schematic instance paths. The generation block 302 may initialize the fuse generation block 108. Next, the repair

memo block 304 may initialize the fuse application block 112. The fuse application block 112 may generate a repair memo. The repair program block 306 may implement the repair memo generated by the repair memo block 304 into a repair program.

The test engineer may run a set of simulations to check the laser repair program. The output of these simulations may be a list of co-ordinates of fuses that need to be blown for the desired repair and the corresponding logical address used. The co-ordinates of the fuse may be stored in the location block 116. The output of the simulation is generally fed back to the design engineer.

The feedback block 310 may allow the Design engineer to feed the sets of co-ordinates from the simulations to the fuse application block 112. The fuse application block 112 may generate a set of verilog statements that may be used to program selected fuses. The map fuse block 312 may map fuse co-ordinates to the verilog program statements stored in the program statements block 114. The logic simulation block 314 may allow the design engineer to run a simulation to check the correctness of the blown fuses utilizing the logical address information.

In case of a discrepancy found in the logic simulation block 314 the SIMS check block 316 may present (i) the repair program to the repair program block 306 or (ii) the repair memo to the repair memo block 304.

In this alternative, essentially the blocks 302, 314, a reversed block 312 are performed first. The particular test cases "sufficient to ensure coverage" are documented in the repair memo block 304. Such documentation may include coordinates (e.g., derived thru an inverse of block 312: program statements are mapped to fuse coordinates (using block 112) and the coordinates are embedded in the memo block 304). The test engineer may then create the repair program as before (e.g., the block 306) from the repair memo block 304. The test engineer may then employ the repair program in a simulate-mode (e.g., in the block 308) to output coordinates from each test case. Next, a compare of these coordinates to the coordinates documented in the repair memo is performed that may decide whether an error exists in the memo or the program. Such a comparison may be similar to the block 316, although the "simulation" being verified is generally very different. For a program error, the blocks 306, 308, and 316 may

be repeated. For a memo error, the blocks 304, 306, 308, and 316 may be repeated.

While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention.